

Spojové struktury

Jan Faigl

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přednáška 08

BAB36PRGA – Programování v C

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

1 / 50

Úvod

Spojový seznam

Start/End

Vložení/odebrání prvku

Kruhový spojový seznam

Obousměrný seznam

Část I

Část 1 – Spojové struktury

Přehled témat

■ Část 1 – Spojové struktury

Spojové struktury

Spojový seznam

Spojový seznam s odkazem na konec seznamu

Vložení/odebrání prvku

Kruhový spojový seznam

Obousměrný seznam

■ Část 2 – Zadání 8. domácího úkolu (HW8)

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

2 / 50

Úvod

Spojový seznam

Start/End

Vložení/odebrání prvku

Kruhový spojový seznam

Obousměrný seznam

Kolekce prvků (položek)

- V programech je velmi běžný požadavek na uchování seznamu (množiny) prvků (proměnných/struktur).
- Základní kolekce je pole.
 - Jedná se o kolekci položek (proměnných) stejného typu.
 - + Umožňuje jednoduchý přístup k položkám indexací prvků.
 - Položky jsou stejného typu (velikosti), kompilátor tak může vytvořit kód, ve kterém se adresa prvku spočítá z indexu a velikosti prvku.*
 - Velikost pole je určena při vytvoření pole.
 - Velikost (maximální velikost) musí být známa v době vytváření.
 - Změna velikost není přímo možná.
 - Nutné nové vytvoření (allokace paměti), voláním `realloc()` může dojít k rozšíření, které závisí na aktuálním stavu paměti.*
 - Využití pouze malé části pole (s objemnými prvky) může být plýtváním paměti.
 - V případě řazení pole přesouváme jednotlivé položky pole.
 - Vložení prvku a vyjmutí prvku vyžaduje kopírování (zachování souvislosti dat).

Kopírování objemných prvků lze případně řešit ukládáním ukazatelů.

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

3 / 50

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

5 / 50

Seznam – list

- Seznam (proměnných nebo objektů) patří mezi základní datové struktury.
Základní **ADT** – Abstract Data Type.
- Seznam zpravidla nabízí sadu základních operací:
 - Vložení prvku (**insert**);
 - Odebrání prvku (**remove**);
 - Vyhledání prvku (**indexOf**);
 - Aktuální počet prvku v seznamu (**size**).
- Implementace seznamu může být založena na poli nebo spojové struktuře.
 - Pole
 - Indexování je velmi rychlé.
 - Vložení prvku na konkrétní pozici může být pomalé.
 - **Spojové seznamy**
 - Položky seznamu jsou sekvenčně propojeny, přímý náhodný přístup není jednoduše možný.
 - Vložení nebo odebrání prvku může být velmi rychlé.

Nová alokace a kopírování.

Základní operace se spojovým seznamem

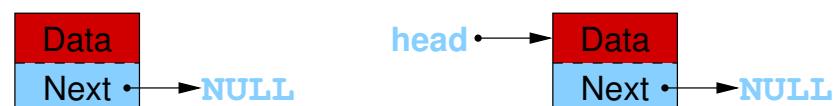
- Vložení prvku:
 - Předchozí prvek odkazuje na nový prvek;
 - Nový prvek může odkazovat na předchozí prvek, který na něj odkazuje.

Tzv. obousměrný spojový seznam.
- Odebrání prvku:
 - Předchozí prvek aktualizuje hodnotu odkazu na následující prvek;
 - Předchozí prvek nově odkazuje na následující prvek, na který odkazoval odebraný prvek.
- Základní implementaci spojového seznamu je
jednosměrný spojový seznam.

Spojové seznamy

- Datová struktura realizující seznam dynamické délky.
- Každý prvek seznamu obsahuje:
 - Datovou část (hodnota proměnné / objekt / ukazatel na data);
 - Odkaz (ukazatel) na další prvek v seznamu.
- První prvek seznamu se zpravidla označuje jako **head** nebo **start**.

Realizujeme jej jako ukazatel odkazující na první prvek seznamu.



Jednosměrný spojový seznam

- Příklad spojového seznamu pro uložení číselných hodnot.
 - Přidání nové hodnoty 50 je přidání nového prvku na konec seznamu.
 - Odebrání prvku s hodnotou 79.
1. Nejdříve sekvenčně najdeme prvek s hodnotou 79.
 2. Následně vyjmeme prvek s hodnotou a propojíme prvek 93 s prvkem 50.
- Položku next prvku 93 nastavíme na hodnotu next odebraného prvku, tj. na následující prvek s hodnotou 50.*

Spojový seznam

- Seznam tvoří struktura prvku s dvěma základními položkami:

- Data prvku (může být ukazatel);
- Odkaz (ukazatel) na další prvek.

- Seznam je pak:

1. Ukazatel na první prvek `head`;
2. nebo vlastní struktura pro seznam.

Vhodné pro uložení dalších informací, počet prvků, poslední prvek.

- Příklad struktur pro uložení spojového seznamu celých čísel.

```
1 typedef struct entry {           1   typedef struct {
2     int value;                  2     entry_t *head;
3     struct entry *next;         3     entry_t *tail;
4 } entry_t;                      4     int counter; // pocet prvků
5 entry_t *head = NULL;            5 } linked_list_t;
```

- Pro jednoduchost prvky seznamu obsahují celé číslo.

Obecně mohou obsahovat libovolná data (ukazatel na strukturu).

Spojový seznam – push()

- Přidání prvku na začátek implementujeme ve funkci `push()`.

- Předáváme adresu, kde je uložen odkaz na start seznamu.

head je ukazatel, proto předáváme adresu proměnné, tj. `&head` a parametr je ukazatel na ukazatel.

```
1 void push(int value, entry_t **head)
2 { // add new entry at front of the list
3     entry_t *new_entry = myMalloc(sizeof(entry_t));
4
5     new_entry->value = value; // set data
6     if (*head == NULL) { // first entry in the list
7         new_entry->next = NULL; // reset the next
8     } else {
9         new_entry->next = *head;
10    }
11    *head = new_entry; //update the head
12 }
```

Alternativně můžeme `push()` implementovat také jako `entry_t push(int value, entry_t *head)`.*

- Přidání prvku není závislé na počtu prvků v seznamu.

Konstantní složitost operace `push()` – $O(1)$.

Přidání prvku – příklad

1. Vytvoříme nový prvek (10) seznamu a uložíme odkaz v `head`.

```
head = myMalloc(sizeof(entry_t));
head->value = 10;
head->next = NULL;
```

2. Další prvek (13) přidáme propojením s aktuálně 1. prvkem.

```
entry_t *new_entry = myMalloc(sizeof(entry_t));
new_entry->value = 13;
new_entry->next = head;
```

3. a aktualizací proměnné `head`.

```
head = new_entry;
```

- Stále máme přístup na všechny prvky přes `head` a `head->next`.

- **Inicializace položek prvku je důležitá.**

- Hodnota `head == NULL` indikuje prázdný seznam.

- Hodnota `entry->next == NULL` indikuje poslední prvek seznamu.

Kontrola dynamické alokace

```
#include <stdlib.h>

void* myMalloc(size_t size)
{
    void *ret = malloc(size);
    if (!ret) {
        fprintf(stderr, "Malloc failed!\n");
        exit(-1)
    }
    return ret;
}
```

lec08/my_malloc.h, lec08/my_malloc.c

Spojový seznam – pop()

- Odebrání prvního prvku ze seznamu

Kdy použijeme assert() a kdy myAssert()?

```
1 int pop(entry_t **head)
2 { // linked list must be non-empty
3     assert(head != NULL && *head != NULL);
4     entry_t *prev_head = *head; // save the current head
5     int ret = prev_head->value; // retrieve data from the current head
6     *head = prev_head->next; // set to NULL if the last item is popped
7     free(prev_head); // release memory of the popped entry
8     return ret;
9 }
```

*Alternativně například také jako `int pop(entry_t *head)`, ale nenastaví `head` na `NULL` v případě vyjmutí posledního prvku.*

- Odebrání prvku není závislé na počtu prvků v seznamu.

Konstantní složitost operace `pop()` – $O(1)$.

Spojový seznam – size()

- Zjištění počtu prvků v seznamu vyžaduje projít seznam až k zarázce `NULL`.

Poslední položka je taková, pro kterou platí `next == NULL`, nebo je seznam prázdný a `head == NULL`.

- Proměnnou `cur` používáme jako „kurzor“ pro procházení seznamu.

```
1 int size(const entry_t *const head)
2 { // const - we do not attempt to modify the list
3     int counter = 0;
4     const entry_t *cur = head;
5     while (cur) { // or cur != NULL
6         cur = cur->next;
7         counter += 1;
8     }
9     return counter;
10 }
```

Použijeme konstantní ukazatel na konstantní proměnnou, neboť seznam pouze procházíme a nemodifikujeme. Z hlavičky funkce je tak zřejmé, že vstupní strukturu ve funkci nemodifikujeme.

- Procházíme kompletní seznam (n prvků), abychom spočítali počet prvků seznamu.

Lineární složitost operace `size()` – $O(n)$.

Spojový seznam – back()

- Vrácení hodnoty posledního prvku ze seznamu – `back()`.

```
1 int back(const entry_t *const head)
2 {
3     const entry_t *end = head;
4     while (end && end->next) { // 1st test list is not empty
5         end = end->next;
6     }
7     assert(end); //do not allow calling back on empty list
8     return end->value;
9 }
```

Kontrolou assert() vynucujeme, že při implementaci programu ladíme, že volání back() nebude probědět pro prázdný seznam. To musíme zajistit programově.

- Musíme projít všechny prvky seznamu.

Lineární složitost operace `back()` – $O(n)$.

Spojový seznam – procházení seznamu

- Procházení seznamu demonstруjeme na funkci `print()`.

```
1 void print(const entry_t *const head)
2 {
3     const entry_t *cur = head; // set the cursor to head
4     while (cur != NULL) {
5         printf("%i%s", cur->value, cur->next ? " " : "\n");
6         cur = cur->next; // move in the linked list
7     }
8 }
```

- Použijeme konstantní ukazatel na konstantní proměnnou, neboť seznam pouze procházíme a nemodifikujeme.

Z hlavičky funkce je zřejmé, že vstupní strukturu nemodifikujeme.

- Prvky seznamu tiskneme za sebou oddělené mezerou a poslední prvek je zakončen znakem nového řádku.

Příklad – Spojový seznam celých čísel

```
entry_t *head;
head = NULL; // initialization is important

push(17, &head);
push(7, &head);
printf("List: ");
print(head);
push(5, &head);
printf("\nList size: %i\n", size(head));
printf("Last entry: %i\n\n", back(head));
printf("List: ");
print(head);
push(13, &head);
push(11, &head);
pop(&head);
printf("List:r");
print(head);
printf("\nPop until head is not empty\n");
while (head != NULL) {
    const int value = pop(&head);
    printf("Popped value %i\n", value);
}
printf("List size: %i\n", size(head));
printf("Last entry value %i\n", back(head));
```

```
$ clang -g demo-linked_list-int.c
linked_list.c
$ ./a.out
List: 7 17
List size: 3
Last entry: 17
List: 5 7 17
List: 13 5 7 17
Cleanup using pop until head is not
empty
Popped value 13
Popped value 5
Popped value 7
Popped value 17
List size: 0
lec08/linked_list-int.h
lec08/linked_list-int.c
lec08/demo-linked_list-int.c
```

Spojový seznam – zrychlení operací `size()` and `back()`

- Operace `size()` a `back()` procházejí kompletní seznam.
- Operaci `size()` můžeme urychlit udržováním aktuálního počtu prvků v seznamu.
 - Zavedeme datovou položku `int counter`.
 - Počet prvků inkrementujeme při každém přidání prvku a dekrementuje při každém odebrání prvku.
- Operaci `back()` můžeme urychlit proměnou odkazující na poslední prvek.
- Zavedeme strukturu pro vlastní spojový seznam s položkami `head`, `counter`, and `tail`.

```

1 typedef struct {
2     entry_t *head;
3     entry_t *tail;
4     int counter;
5 } linked_list_t;

```

■ V případě přidání prvku na začátek, aktualizujeme `tail` pouze pokud byl seznam dosud prázdný.
 ■ Proměnnou `tail` aktualizujeme při přidání prvku na konec nebo vyjmutí posledního prvku.

Spojový seznam – `push()` s odkazem na konec seznamu

```

1 void push(int value, linked_list_t *list)
2 { // add new entry at front
3     assert(list);
4     entry_t *new_entry = myMalloc(sizeof(entry_t));
5     new_entry->value = value; // set data; exit is called if myMalloc fails
6     if (list->head) { // an entry already in the list
7         new_entry->next = list->head;
8     } else { //list is empty
9         new_entry->next = NULL; // reset the next
10    list->tail = new_entry; //1st entry is the tail
11 }
12 list->head = new_entry; //update the head
13 list->counter += 1; // keep counter up to date
14 }

Hodnotu ukazatele tail nastavujeme pouze pokud byl seznam prázdný, protože prvky přidáváme na začátek.

```

Spojový seznam – urychlený `size()`

- Samostatná struktura pro seznam.
 - Položky `head` a `counter`.
 - `head` je ukazatel na `entry_t`.
 - Ve funkce `size()` předpokládáme validní odkaz na seznam.
 - Proto voláme `assert(list)`.
 - Přímá inicializace `linked_list_t linked_list = { NULL, 0 }`;
 - Do funkcí `push()` a `pop()` stačí předávat pouze ukazatel, proto použijeme proměnnou `list linked_list_t *list = &linked_list;`
 - Inkrementujeme a dekrementujeme proměnnou `counter` ve funkích `push()` a `pop()`.
- ```

void push(int data, linked_list_t *list)
{
 ...
 list->counter += 1;
}

int pop(linked_list_t *list)
{
 ...
 list->counter -= 1;
 return ret;
}

```

## Spojový seznam – `pop()` s odkazem na konec seznamu

- Při volání musí být odkaz na spojový seznam platný (nikoliv `NULL`).
  - `assert()` testujeme správnost volání, že jsme ve struktuře programu neudělali chybu. Po odladění můžeme test vypustit, např. `NDEBUG`.
  - `myAssert()` testuje, že data jsou za běhu programu správně. Pokud ne, ukončujeme program a reportujeme.
 

V našem konkrétním případě můžeme také zajistit programově použitím `assert()` a podmínit volání `pop()`, např. `if (!is_empty(list))`.
- ```

1 int pop(linked_list_t *list)
2 {
3     assert(list);
4     myAssert(list->head, __LINE__, __FILE__); // non-empty list
5     entry_t *prev_head = list->head; // save head
6     list->head = prev_head->next;
7     list->counter -= 1; // keep counter up to date
8     int ret = prev_head->value;
9     free(prev_head); // release the memory
10    if (list->head == NULL) { // end has been popped
11        list->tail = NULL;
12    }
13    return ret;      Hodnotu proměnné tail nastavujeme pouze pokud byl odebrán poslední prvek, protože
14 }                  prvky odebíráme ze začátku.

```

```

myAssert()
1 #ifndef __MY_ASSERT_H__
2 #define __MY_ASSERT_H__
4 #include <stdio.h> //fprintf()
5 #include <stdlib.h> //exit() and malloc()
7 #define myAssert(x, line, file) \
8     if (!(x)) { \
9         fprintf(stderr, "my_assert fail, \
10             line: %d, file %s\n", line, file); \
11         exit(-1); \
12     }

```

Výpis chyby s číslem řádku a jménem zdrojového souboru pro rychlejší nalezení kontextu a případnou opravu.

Spojový seznam – back() s odkazem na konec seznamu

- Proměnná **tail** je buď **NULL** nebo odkazuje na poslední prvek seznamu.

```
1 int back(const linked_list_t *const list)
2 { // const - we do not modify the linked list
3   // we do not allow to call back on empty list that has to be
4   // assured programmatically
5   assert(list && list->tail);
6   return list->tail->value;
7 }
```

- Udržováním hodnoty proměnné **tail** (ve funkčích **push()** a **pop()**) jsme snížili časovou náročnost operace **back()** z lineární složitosti na počtu prvků (n) v seznamu $O(n)$ na konstantní složitost $O(1)$.

Spojový seznamu – popEnd()

- Odebrání prvku z konce seznamu.

```
1 int popEnd(linked_list_t *list)
2 {
3   assert(list && list->head);
4   entry_t *end = list->tail; // save the end
5   if (list->head == list->tail) { // the last entry is
6     list->head = list->tail = NULL; // removed
7   } else { // there is also penultimate entry
8     entry_t *cur = list->head; // that needs to be
9     while (cur->next != end) { // updated (its next
10       cur = cur->next; // pointer to the next entry
11     }
12     list->tail = cur;
13     list->tail->next = NULL; //the tail does not have next
14   }
15   int ret = tail->value;
16   free(end);
17   list->counter -= 1;
18   return ret;      Složitost je  $O(n)$ , protože musíme aktualizovat předposlední prvek. Alternativně
19 }                  lze řešit obousměrným spojovým seznamem.
```

Spojový seznamu – pushEnd()

- Přidání prvku na konec seznamu.

```
1 void pushEnd(int value, linked_list_t *list)
2 {
3   assert(list);
4   entry_t *new_entry = myMalloc(sizeof(entry_t));
5   new_entry->value = value; // set data
6   new_entry->next = NULL; // set the next
7   if (list->tail == NULL) { //adding the 1st entry
8     list->head = list->tail = new_entry;
9   } else {
10    list->tail->next = new_entry; //update the current tail
11    list->tail = new_entry;
12  }
13  list->counter += 1;
14 }
```

- Na asymptotické složitost metody přidání dalšího prvku (na konec seznamu) se nic nemění, je nezávislé na aktuálním počtu prvků v seznamu.

Příklad použití

- Příklad použití na seznam hodnot typu **int**.

```
1 #include "linked_list.h"
3 linked_list_t list = { NULL, NULL, 0 };
4 linked_list_t *lst = &list;
5 push(10, lst); push(5, lst); pushEnd(17, lst);
6 push(7, lst); pushEnd(21, lst);
7 print(lst);
9 printf("Pop 1st entry: %i\n", pop(lst));
10 printf("Lst: "); print(lst);
12 printf("Back of the list: %i\n", back(lst));
13 printf("Pop from the end: %i\n", popEnd(lst));
14 printf("Lst: "); print(lst);
16 free_list(lst); // cleanup!!!
```

- Výstup programu

```
$ clang linked_list.c demo-linked_list.c &&
  ./a.out
7 5 10 17 21
Pop 1st entry: 7
Lst: 5 10 17 21
Back of the list: 21
Pop from the end: 21
Lst: 5 10 17
```

lec08/linked_list.h
lec08/linked_list.c
lec08/demo-linked_list.c

Spojový seznam – Vložení prvku do seznamu

- Vložení do seznamu:
 - na začátek – modifikujeme proměnnou **head** (funkce `push()`);
 - na konec – modifikujeme proměnnou posledního prvku a nastavujeme nový konec **tail** (funkce `pushEnd()`);
 - obecně – potřebujeme prvek (**entry**), za který chceme nový prvek (`new_entry`) vložit.

```
entry_t *new_entry = myMalloc(sizeof(entry_t));
new_entry->value = value; // nastavení hodnoty
new_entry->next = entry->next; // propojení s nasledujicim
entry->next = new_entry; // propojení entry
```

- Do seznamu můžeme chtít prvek vložit na konkrétní pozici, tj. podle indexu v seznamu.
Případně můžeme také požadovat vložení podle hodnoty prvku, tj. vložit před prvek s příslušnou hodnotou. Např. vložením prvku vždy před první prvek, který je větší vytvoříme uspořádany seznam – realizujeme tak řazení vkládáním (insert sort).
- Nalezení první implementujeme „privátní“ (**static**) funkcí `getEntry()`.

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

29 / 50

Spojový seznam – insertAt()

- Vložení nového prvku na pozici `index` v seznamu.
- ```
void insertAt(int value, int index, linked_list_t *list)
{
 assert(list); // list != NULL
 if (index < 0) { return; } // only positive position
 if (index == 0) { // handle the 1st position
 push(value, list);
 return;
 }
 entry_t *new_entry = myMalloc(sizeof(entry_t));
 new_entry->value = value; // set data
 entry_t *entry = getEntry(index - 1, list);
 if (entry != NULL) { // entry can be NULL for the 1st
 new_entry->next = entry->next; // entry (empty list)
 entry->next = new_entry;
 }
 if (entry == list->tail) {
 list->tail = new_entry; // update the tail
 }
 list->counter += 1; // Pro napojení spojového seznamu potřebuje položku next, proto hledáme prvek na pozici (index - 1)-getEntry().
}
```

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

31 / 50

## Spojový seznam – getEntry()

- Nalezení prvku na pozici `index`.
  - Pokud je `index` větší než počet prvků v poli, návrat posledního prvku.
- ```
static entry_t* getEntry(int index, const linked_list_t *list)
{
    // here, we assume index >= 0
    entry_t *cur = list->head;
    int i = 0;
    while (i < index && cur != NULL && cur->next != NULL) {
        cur = cur->next;
        i += 1;
    }
    return cur; //return entry at the index or the last entry
}
```
- Pokud je seznam prázdný vrátí `NULL`, tj. `list->head == NULL`.*
- Funkci `getEntry()` chceme používat privátně pouze v rámci jednoho modulu (`linked_list.c`).
 - Proto ji definujeme s modifikátorem **static**.

Viz lec08/linked_list.c

Příklad vložení prvků do seznamu – insertAt()

- Příklad vložení do seznam čísel
- ```
linked_list_t list = { NULL, NULL, 0 };
linked_list_t *lst = &list;

push(10, lst); push(5, lst); push(17, lst);
push(7, lst); push(21, lst);
print(lst);

insertAt(55, 2, lst);
print(lst);

insertAt(0, 0, lst);
print(lst);

insertAt(100, 10, lst);
print(lst);

free_list(lst); // cleanup!!!

```

- Výstup programu
- ```
$ clang linked_list.c demo-insertat.c
  && ./a.out
21 7 17 5 10
21 7 55 17 5 10
0 7 55 17 5 10
0 7 55 17 5 10 100
```

lec08/demo-insertat.c

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

32 / 50

Spojový seznam – removeAt(int index)

- Odebrání prvku na pozici `int index` a navázání seznamu.
- Pokud `index > size - 1`, smaže poslední prvek, viz `getEntry()`.
- Pro navázání seznamu potřebujeme prvek na pozici `index - 1`.

```
void removeAt(int index, linked_list_t *list)
{
    // check the arguments first
    if (index < 0 || list == NULL || list->head == NULL) { return; }
    if (index == 0) {
        pop(list);
    } else {
        entry_t *entry_prev = getEntry(index - 1, list);
        entry_t *entry = entry_prev->next;
        if (entry != NULL) { //handle connection
            entry_prev->next = entry_prev->next->next;
        }
        if (entry == list->tail) {
            list->tail = entry_prev;
        }
        free(entry);
        list->count -= 1;
    }
}
```

Složitost v nejnepříznivějším případě $O(n)$, protože nejdříve musíme prvek najít.

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

33 / 50

Příklad spojový seznamu textových řetězců

- Je nutné zvolit přístup pro alokaci hodnot textových řetězců.
- Literály vs. proměnné.
- Příklad použití.

V `lec08/linked_list-str.c` je zvolena **alokace paměti a kopírování hodnot**.

```
#include "linked_list-str.h"
linked_list_t list = { NULL };
linked_list_t *lst = &list;

push("FEE", lst); push("CTU", lst);
push("PRG", lst); push("Lec08", lst);
print(lst);

for (int i = 0; i < 2; ++i) {
    char *str = pop(lst);
    printf("%d. %s\n", i+1, str);
    free(str); // Free is needed!
}

$ ./demo-linked_list-str
Lec08 PRG CTU FEE
1. popped string Lec08
2. popped string PRG
Popped value "CTU"
String of the popped value is at address 0
x80024c010
Due to selected implementation the memory must
be explicitly deallocated!

```

lec08/demo-linked_list-str.c

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

35 / 50

Příklad použití removeAt(int index)

```
void removeAndPrint(int index, linked_list_t *lst) {
    entry_t* e = getAt(index, lst);
    printf("Remove entry at %i (%i)\n", index,
           e ? e->value : -1);
    removeAt(index, lst);
    print(lst);
}

linked_list_t list = { NULL, NULL, 0 };
linked_list_t *lst = &list;
push(10, lst); push(5, lst); push(17, lst); push
(7, lst); push(21, lst);
print(lst);
removeAndPrint(3, lst);
removeAndPrint(3, lst);
removeAndPrint(0, lst);
free_list(lst); // cleanup!!!

```

lec08/demo-removeat.c

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

34 / 50

Spojový seznam s hodnotami typu textový řetězec

- Zajištění správné alokace a uvolnění paměti je v našem příkladě náročnější.
- V případě volání `pop()` je nutné následně paměť uvolnit.

V C++ lze řešit například prostřednictvím „smart pointers“.

```
/* WARNING printf("Popped value \"%s\"\n", pop(lst)); */
/* Note, using this will cause memory leakage since we lost the address
   value to free the memory!!! */

char *str = pop(lst);
printf("Popped value \"%s\"\n", str);
free(str); /* str must be deallocated */

```

Při práci s dynamickou pamětí a datovými strukturami je nutné zvolit vhodný model (např. kopírování dat) a zajistit správné uvolnění paměti.

- Podobně jako textové řetězce se bude chovat ukazatel na nějakou komplexnější strukturu.
- Projděte si přiložené příklady, zkuste si naimplementovat vlastní řešení a otestovat správnou alokaci a uvolnění paměti!**

lec08/linked_list-str.h, lec08/linked_list-str.c, lec08/demo-linked_list-str.c

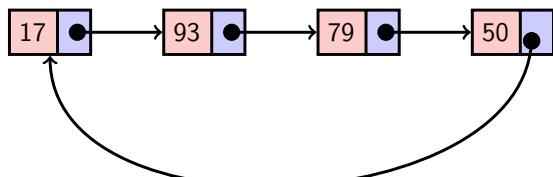
BAB36PRGA – Přednáška 08: Spojové struktury

36 / 50

Jan Faigl, 2024

Kruhový spojový seznam

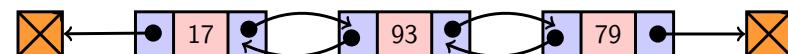
- Položka `next` posledního prvku může odkazovat na první prvek.
- Tak vznikne kruhový spojový seznam.



- Při přidání prvku na začátek je nutné aktualizovat hodnotu `next` posledního prvku.

Obousměrný spojový seznam

- Každý prvek obsahuje odkaz na následující a předchozí položku v seznamu, položky `prev` a `next`.
- První prvek má nastavenou položku `prev` na hodnotu `NULL`.
- Poslední prvek má `next` nastaveno na `NULL`.
- Příklad obousměrného seznamu celých čísel.



Příklad – Obousměrný spojový seznam

- Prvek listu má hodnotu (`value`) a dva odkazy (`prev` a `next`).
- Alokaci prvku provedeme funkcí s inicializací na základní hodnoty.

```

typedef struct dll_entry {
    int value;
    struct dll_entry *prev;
    struct dll_entry *next;
} dll_entry_t;

typedef struct {
    dll_entry_t *head;
    dll_entry_t *tail;
} doubly_linked_list_t;

dll_entry_t* allocate_dll_entry(int value)
{
    dll_entry_t *new_entry = myMalloc(sizeof(dll_entry_t));
    new_entry->value = value;
    new_entry->next = NULL;
    new_entry->prev = NULL;
    return new_entry;
}

lec08/doubly_linked_list.h, lec08/doubly_linked_list.c
  
```

Obousměrný spojový seznam – vložení prvku

- Vložení prvku před prvek `cur`:

 - Napojení vloženého prvku do seznamu, hodnoty `prev` a `next`;
 - Aktualizace `next` předchozího prvku k prvku `cur`;
 - Aktualizace `prev` proměnného prvku `cur`.

```

void insert_dll(int value, dll_entry_t *cur)
{
    assert(cur);
    dll_entry_t *new_entry = allocate_dll_entry(value);
    new_entry->next = cur;
    new_entry->prev = cur->prev;
    if (cur->prev != NULL) {
        cur->prev->next = new_entry;
    }
    cur->prev = new_entry;
}
  
```

Obousměrný spojový seznam – přidání prvku na začátek push()

```
void push_dll(int value, doubly_linked_list_t *list)
{
    assert(list);
    dll_entry_t *new_entry = allocate_dll_entry(value);
    if (list->head) { // an entry already in the list
        new_entry->next = list->head; // connect new -> head
        list->head->prev = new_entry; // connect new <- head
    } else { //list is empty
        list->tail = new_entry;
    }
    list->head = new_entry; //update the head
}
```

lec08/doubly_linked_list.c

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

43 / 50

Příklad použití

```
#include "doubly_linked_list.h"

doubly_linked_list_t list = { NULL, NULL };
doubly_linked_list_t *lst = &list;

push_dll(17, lst); push_dll(93, lst);
push_dll(79, lst); push_dll(11, lst);

printf("Regular print: ");
print_dll(lst);

printf("Revert print: ");
printReverse(lst);

free_dll(lst);
```

lec08/doubly_linked_list.c
lec08/demo-doubly_linked_list.c

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

45 / 50

Obousměrný spojový seznam – tisk seznamu

```
void print_dll(const doubly_linked_list_t *list)
{
    if (list && list->head) {
        dll_entry_t *cur = list->head;
        while (cur) {
            printf("%i%s", cur->value, cur->next ? " " : "\n");
            cur = cur->next;
        }
    }
}

void printReverse(const doubly_linked_list_t *list)
{
    if (list && list->tail) {
        dll_entry_t *cur = list->tail;
        while (cur) {
            printf("%i%s", cur->value, cur->prev? " " : "\n");
            cur = cur->prev;
        }
    }
}
```

lec08/doubly_linked_list.c

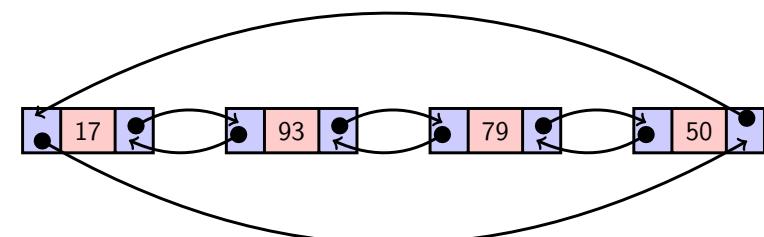
Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

44 / 50

Kruhový obousměrný seznam

- Výstup programu
- \$ clang doubly_linked_list.c
demo-double_linked_list.c
\$./a.out
- Regular print: 11 79 93 17
Revert print: 17 93 79 11
- Položka `next` posledního prvku odkazuje na první prvek.
- Položka `prev` prvního prvku odkazuje na poslední prvek.



Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

46 / 50

Část II

Část 2 – Zadání 8. domácího úkolu (HW8)

Shrnutí přednášky

Zadání 8. domácího úkolu HW8

Téma: Fronta spojovým seznamem s řazením

Povinné zadání: **3b**; Volitelné zadání: **4b**; Bonusové zadání: **není**

- **Motivace:** Práce s pamětí a datovými strukturami.
- **Cíl:** Prohloubit si znalost paměťové reprezentace a dynamické alokace při práci se spojovým seznamem.
- **Zadání:** <https://cw.fel.cvut.cz/wiki/courses/bab36prga/hw/hw8>
 - Implementace spojového seznamu s vkládáním a odebíráním prvků.
 - Volitelné zadání rozšiřuje úlohu o vkládáníím prvků s řazením.
- Termín odevzdání: **11.05.2024, 23:59:59 PDT.**

Diskutovaná témata

- Spojové struktury
 - Jednosměrný spojový seznam;
 - Obousměrný spojový seznam;
 - Kruhový obousměrný spojový seznam.
- Implementace operací `push()`, `pop()`, `size()`, `back()`, `pushEnd()`, `popEnd()`,
`insertAt()`, `getEntry()`, `getAt()`, `removeAt()`, `indexOf()`.
- Použití spojového seznamu pro dynamicky alokované hodnoty prvků seznamu.
- **Příště abstraktní datový typ (ADT).**

Část IV

Appendix

Kódovací příklad – list.c 1/2

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4
5 #include "list.h"
6
7 struct item {
8     void *value;
9     struct item *next;
10};
11
12 struct list {
13     struct item *root;
14     bool (*isLess)(const void *a, const void *b);
15};
16
17 enum { ERROR_MEM = 101 };
18
19 void* create(void)
20 {
21     struct list *ret = malloc(sizeof(struct list));
22     if (!ret) {
23         fprintf(stderr, "ERROR: cannot allocate memory\n");
24         exit(ERROR_MEM);
25     }
26     ret->root = NULL;
27     ret->isLess = NULL;
28     return ret;
29 }
30
31 void release(void** list)
32 {
33     if (list && *list) {
34         struct list *lst = (struct list*)list;
35         struct item *cur = lst->root;
36         while (cur) {
37             struct item *t = cur;
38             cur = cur->next;
39             free(t->value); //Případě specifická funkce
40             free(t); // pro složený typ s ukazateli
41         }
42         free(*list);
43         *list = NULL;
44     }
45 }
```

- Při chybě alokace program končí voláním `exit()` s návratovou hodnotou 101.
- Definici složených typů implementujeme pouze v souboru `list.c`, může se případně v budoucnu měnit, proto není `struct item` součást rozhraní v `list.h`.

Kódovací příklad – Dynamická knihovna

- Knihovna s implementací (spojového) seznamu pro ukládání dynamicky alokovaných položek.
- Pro jednoduchost při chybě dynamické alokace program ukončíme s výstupem na `stderr`.
- Implementujeme funkci `push()`, která nepřidá hodnoty `NULL`. Návrhová volba!
- Prázdný seznam je indikován návratovou hodnotou `NULL` funkce `pop()`.
- Implementujeme nastavení funkce porovnání položek `setLess()`, kterou využijeme při vkládání položek do seznamu. Uspořádání položek dojde při volání `push()`. Implementujeme `insert sort`.
- Vytvoříme dvě verze knihovny s/bez uspořádání položek, které budeme linkovat dynamicky.

```

1 #ifndef __LIST_H__
2 #define __LIST_H__
3
4 void* create(void); // void* - konkrétní typ považujeme za vnitřní záležitost knihovny.
5 void release(void** list); // argument list musí odpovídat typu z volání create()!
6
7 void setLess(void *list, bool (*isLess)(const void *a, const void *b));
8
9 void push(void *list, void *value);
10 void* pop(void *list);
11
12 #endif
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
777
778
779
779
780
781
782
783
784
785
785
786
787
788
789
789
790
791
792
793
794
795
795
796
797
798
799
799
800
801
802
803
803
804
805
806
807
808
809
809
810
811
812
813
814
815
815
816
817
818
819
819
820
821
822
823
824
825
825
826
827
828
829
829
830
831
832
833
834
834
835
836
837
838
838
839
839
840
841
842
843
843
844
845
846
846
847
847
848
849
849
850
851
852
852
853
853
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1
```

Kódovací příklad – list.c 3/3

```

66 static void pushLess(struct list *list, struct item *item)
67 {
68     if (!list->root || list->isLess(item->value, list->root->
69         value)) {
70         item->next = list->root; // ok i pro root == NULL
71         list->root = item;
72         return;
73     }
74
75     struct item *prev = list->root;
76     struct item *cur = prev->next; // list->root není NULL
77     while (cur && !list->isLess(item->value, cur->value)) {
78         prev = cur;
79         cur = cur->next;
80     }
81     item->next = cur; //item bude poslední if cur == NULL
82     prev->next = item;
83 }
84 
```

- Prvátní funkce `pushLess()` v rámci `list.c`.
- Funkce využívá nastavenou funkci `list->isLess`.
- Funkce vkládá `item` před první položku seznamu, která je větší (dle `isLess()`).

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

56 / 50

```

85 void push(void *list, void *value)
86 {
87     struct list *lst = (struct list*)list;
88     if (!lst || !value) { // ukládání hodnot NULL
89         return; // není podporováno (ignorujeme)
90     }
91
92     struct item *n = allocate_item(value);
93     #ifdef WITH_LESS // isLess pouze pokud WITH_LESS
94     if (lst->isLess) {
95         pushLess(lst, n);
96         return;
97     }
98     if (lst->root) {
99         n->next = lst->root;
100    }
101   lst->root = n;
102 }
```

`list.c`

- Výchozí implementace `push()` vkládá hodnoty na začátek seznamu.
- Pokud je `list.c` komplikován s `WITH_LESS`, dochází k využití `isLess()`.

Např. clang -DWITH_LESS list.c

Kódovací příklad – Volání rozhraní seznamu 2/3

```

$ clang -fPIC -c list.c
$ clang -shared -o liblist.so.1 list.o

$ clang -g -DWITH_LESS -fPIC -c list.c
$ clang -shared -o liblist.so.2 list.o

$ ln -s liblist.so.1 liblist.so

$ clang -g -L., -Wl,-rpath=. -llist -o demo demo.c

$ ldd demo
demo:
 liblist.so => ./liblist.so (0x80024d000)
 libc.so.7 => /lib/libc.so.7 (0x800251000)

■ Vytvoříme dvě verze (bez/s isLess()) dynamicky linkované knihovny liblist.so.1 a liblist.so.2.
■ Konkrétní (aktuální verzi) odkážeme symbolickým odkažadem (nebo jen nakopírujeme) jako soubor liblist.so.
■ Program demo.c zkompilujeme s knihovnou list.
■ Dynamicky linkované knihovny programu můžeme zobrazit, např. nástrojem ldd.
        ldd – list dynamic object dependencies.

```

- Vytvoříme vstupní soubor `in.txt` a přesměrujeme `stdin`.
 - cat in.txt
 - 4
 - 2
 - 16
 - 13
 - 6
 - 1
 - 3
 - 5
 - 9
 - 15
 - ./demo <in.txt 2>&1 | head -12
 - DEBUG: read "4"
 - DEBUG: read "2"
 - DEBUG: read "16"
 - DEBUG: read "13"
 - DEBUG: read "6"
 - DEBUG: read "1"
 - DEBUG: read "3"
 - DEBUG: read "5"
 - DEBUG: read "9"
 - DEBUG: read "15"
 - Popped value is "15"
 - Popped value is "9"
 - Popped value is "5"
 - Popped value is "3"
 - Popped value is "1"
 - Popped value is "6"
 - Popped value is "13"
 - Popped value is "16"
 - Popped value is "2"
 - Popped value is "4"
 - Popped value is "5"
 - Popped value is "6"
 - Popped value is "9"

BAB36PRGA – Přednáška 08: Spojové struktury

58 / 50

Kódovací příklad – Volání rozhraní seznamu 1/3

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4 #include <string.h>
5
6 #include "list.h"
7
8 bool isLess(const void *a, const void *b);
9
10 int main(void)
11 {
12     int ret = EXIT_SUCCESS;
13     char *line = NULL; // musí být NULL, alokace v getline()
14     size_t linecap = 0; // délka alokované paměti v getline()
15     ssize_t ln = 0; // délka načteného řetězce
16     void *list = create(); // vytvoření seznamu z list.h
17
18     setLess(list, isLess); // nastavení funkce isLess() demo.c
19
20 }
```

```

21     while ((ln = getline(&line, &linecap, stdin)) > 0) {
22         if (line[ln-1] == '\n') { // ln vždy alespoň 1!
23             line[ln-1] = '\0'; // ignorujeme konec řádku
24         }
25         fprintf(stderr, "DEBUG: read \"%s\"\n", line);
26         push(list, line); // přidáme do seznamu
27         linecap = 0;
28         line = NULL; // vynucujeme novou alokaci
29     }
30     while ((line = pop(list))) {
31         printf("Popped value is \"%s\"\n", line);
32         free(line); // uvolňujeme řádek z paměti
33     }
34     release(&list);
35     return ret;
36 }
37
38 bool isLess(const void *a, const void *b)
39 {
40     return strcmp(a, b) < 0; // lexikografické porovnání
41 }
```

- Přidáním řádku do seznamu delegujeme zodpovědnost za správu paměti (smazání) seznamu, nebo následnému volání `pop()` a smazání řádku.
- Obsah seznamu vytiskneme voláním `pop()`.

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

57 / 50

Kódovací příklad – Volání rozhraní seznamu 3/3

- Verze bez `isLess()`, knihovna `liblist.so.1`.

```

$ rm -rf liblist.so
$ ln -s liblist.so.1 liblist.so
$ ls -l liblist.so
lrwxr-xr-x 1 liblist.so -> liblist.so.1

$ ./demo < in.txt 2>/dev/null
Popped value is "15"
Popped value is "9"
Popped value is "5"
Popped value is "3"
Popped value is "1"
Popped value is "6"
Popped value is "13"
Popped value is "16"
Popped value is "2"
Popped value is "4"
Popped value is "5"
Popped value is "6"
Popped value is "9"
```

- Verze s `isLess()`, knihovna `liblist.so.2`.

```

$ rm -rf liblist.so
$ ln -s liblist.so.2 liblist.so
$ ls -l liblist.so
lrwxr-xr-x 1 liblist.so -> liblist.so.2

$ ./demo < in.txt 2>/dev/null
Popped value is "1"
Popped value is "13"
Popped value is "15"
Popped value is "16"
Popped value is "2"
Popped value is "3"
Popped value is "4"
Popped value is "5"
Popped value is "6"
Popped value is "9"
```

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

59 / 50